DTIC FILE COPY

(1)

50272-101

REPORT DOCUMENTATION PAGE	1. REPORT NO. DCA/SW/MT-88/001q	2.	3. Recipient's Accession No.
4. Title and Subtitle: Defense Communications Agency Upper Level Protocol Test System Transmission Control Protocol Remote Driver Specification			5. Report Date: May 1988 6.
7. Author(s)			8. Performing Organization Rept. No.
9. Performing Organization Name and Address: Defense Communications Agency, Defense Communications Engineering Center, Code R640, 1860 Wiehle Ave., Reston, VA 22090-5500			10. Project/Task/Work Unit No. 11. Contract(C) or Grant(G) No. (C) (G)
12. Sponsoring Organization Name and Address			13. Type of Report & Period Covered: FINAL 14.

15. Supplementary Notes

For magnetic tape, see: ADA195128

AD-A195 135

Abstract (Limit: 200 words)

This document is part of a software package that provides the capability to conformance test the Department of Defense suite of upper level protocols including: Internet Protocol (IP) Mil-Std 1777, Transmission Control Protocol (TCP) Mil-Std 1778, File Transfer Protocol (FTP) Mil-Std 1780, Simple Mail Transfer Protocol (SMTP) Mil-Std 1781 and TELNET Protocol Mil-Std 1782.

See p 1

17. Document Analysis a. Descriptors

Protocol Test Systems
Conformance Testing
Department of Defense Protocol Suite

b. Identifiers/Open-Ended Terms

Internet Protocol (IP)
Transmission Control Protocol (TCP)
File Transfer Protocol (FTP)
Simple Mail Transfer Protocol (SMTP)
TELNET Protocol

c. COSATI Field/Group

18. Availability Statement	19. Security Class (This Report)	21. No. of Pages
Unlimited Release	UNCLASSIFIED	57
	20. Security Class (This Page) UNCLASSIFIED	22. Price

(See ANSI-Z39.18) See Instructions on Reverse

OPTIONAL FORM 272 (4-77)
(Formerly NTIS-35)
Department of Commerce

DEFENSE COMMUNICATIONS AGENCY

UPPER LEVEL PROTOCOL TEST SYSTEM

TRANSMISSION CONTROL PROTOCOL MIL-STD 1778
REMOTE DRIVER SPECIFICATION

MAY 1988

Disclaimer Concerning Warranty and Liability

This software product and documentation and all future updates to it are provided by the United States Government and the Defense Communications Agency (DCA) for the intended purpose of conducting conformance tests for the DoD suite of higher level protocols. DCA has performed a review and analysis of the product along with tests aimed at insuring the quality of the product, but does not warranty or make any claim as to the quality of this product. The product is provided "as is" without warranty of any kind, either expressed or implied. The user and any potential third parties accept the entire risk for the use, selection, quality, results, and performance of the product and updates. Should the product or updates prove to be defective, inadequate to perform the required tasks, or misrepresented, the resultant damage and any liability or expenses incurred as a result thereof must be borne by the user and/or any third parties involved, but not by the United States Government, including the Department of Commerce and/or The Defense Communications Agency and/or any of their employees or contractors.

Distribution and Copyright

Comments

Comments or questions about this software product and documentation can be addressed in writing to:

DCA Code R640
1860 Wiehle Ave
Reston, VA 22090-5500
ATTN: Protocol Test System Administrator

TABLE OF CONTENTS

TABLE OF CONTENTS (Cont'd.)

TABLE OF CONTENTS (Cont'd.)

LIST OF APPENDICES

LIST OF FIGURES

LIST OF TABLES

SECTION 1 - SCOPE AND PURPOSE

This specification describes the Transmission Control Protocol (TCP) Remote Driver for the Protocol Test System. Section 2, "The Protocol Test System," summarizes the testing procedures used by the system. Section 3, "Remote Driver Functions," contains guidelines for developing a TCP Remote Driver where an implementation under test (IUT) resides. In Section 4 a detailed description is given of the commands the Remote Driver must be able to interpret and carry out to perform the TCP testing. Section 5 specifies the response packets the Remote Driver must send to the Central Driver. In Section 6 Remote Driver timing considerations are discussed. Section 7 describes flexibility requirements.

SECTION 2 - THE PROTOCOL TEST SYSTEM

The Protocol Test System (PTS) exercises an IUT performing peer protocol exchanges with a reference implementation at the Protocol Test System site. A driver controls each peer protocol through its upper level interface. To generate reproducible results, a script controls each driver.

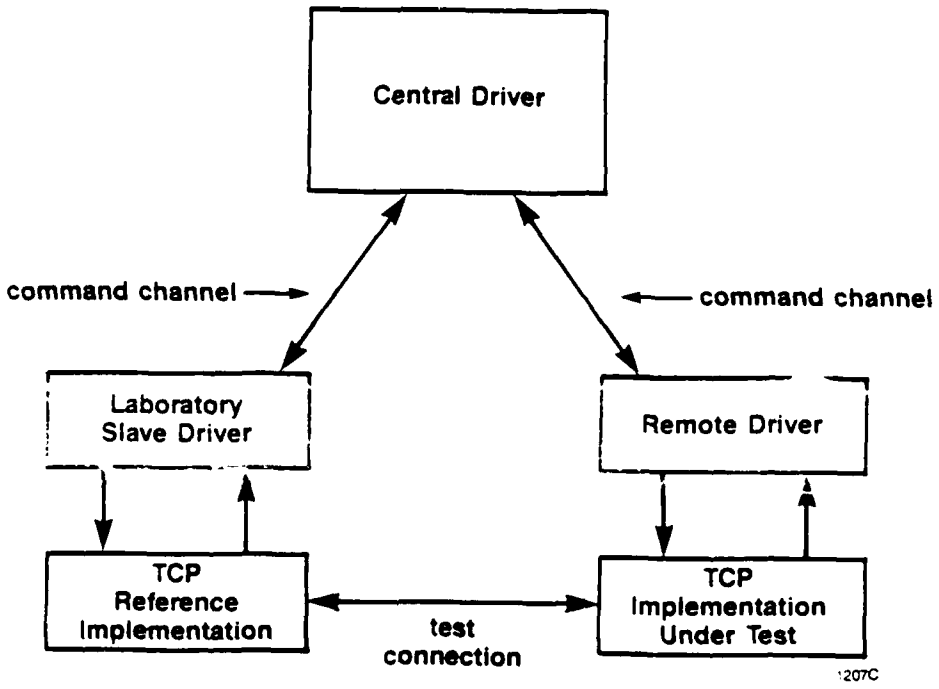
Figure 2-1 illustrates the major components of the Protocol Test System:

- Central Driver (CD) -- To coordinate and monitor protocol testing;
- Remote Driver (RD) -- To exercise the TCP IUT and provide communication links with the Central Driver; and
- Laboratory Slave Driver (LSD) -- To exercise the TCP Reference implementation, record TCP segments exchanged over the test connections, and provide communication links with the Central Driver.

The Central Driver, which coordinates and monitors protocol testing, combines the input it receives from the Laboratory Slave Driver and the Remote Driver to determine the success or failure of each test.

The Remote Driver receives protocol commands from the Central Driver, issues TCP commands to the IUT, and sends back protocol responses from the IUT to the Central Driver. It also executes driver commands received from the Central Driver. Driver commands control the Remote Driver itself and do not affect the IUT's state. Protocol commands that the Remote Driver passes to the IUT as TCP commands, have appropriate parameters and directly affect the IUT's state.

Figure 2-1. Flow of Commands Between the Drivers



SECTION 3 - REMOTE DRIVER FUNCTIONS

The following sections describe how the Remote Driver and Central Driver communicate protocol commands and responses in testing.

3.1 THE COMMAND CHANNEL

All drivers except the Remote Driver reside at the Protocol Test System testing facility. The Remote Driver operates using a transport-level protocol connection as a command channel. The connection links the Remote Driver at its remote site with the Central Driver. This approach to testing assumes the TCP IUT mechanism to be capable of error-free, basic communication. The tester should then be able to rely on the command channel for the simple request/response function required to perform the tests. If not, the first phase of TCP testing will discover the inadequacy, and the implementation will be rejected without further testing.

To set up the command channel, the TCP Remote Driver issues a passive open on port 2001, using default values of the other parameters of passive open (Table 4.1-2). The Remote Driver then waits for the Central Driver to perform an active open to that port. Figure 3.1-1 is a diagram of connection establishment. The Remote Driver must function as a server and listen for further connection requests from the Central Driver on port 2001 after the command channel has been established. Appendix C includes an example of command channel establishment implemented in UNIX/C.

3.2 FLOW OF COMMANDS

Packets containing command codes travel over the command channel from the Central Driver to the Remote Driver. The Remote Driver translates these codes into the appropriate commands, then issues the commands. Similarly, responses from the IUT protocol are received by the Remote Driver and forwarded over the command channel to the Central Driver (Figure 3.2-1). The Central Driver determines from the test responses whether the IUT has functioned as expected and reacts accordingly--by taking the next appropriate action in the testing process--and the flow of commands is repeated.

Figure 3.1-1. Connection Establishment

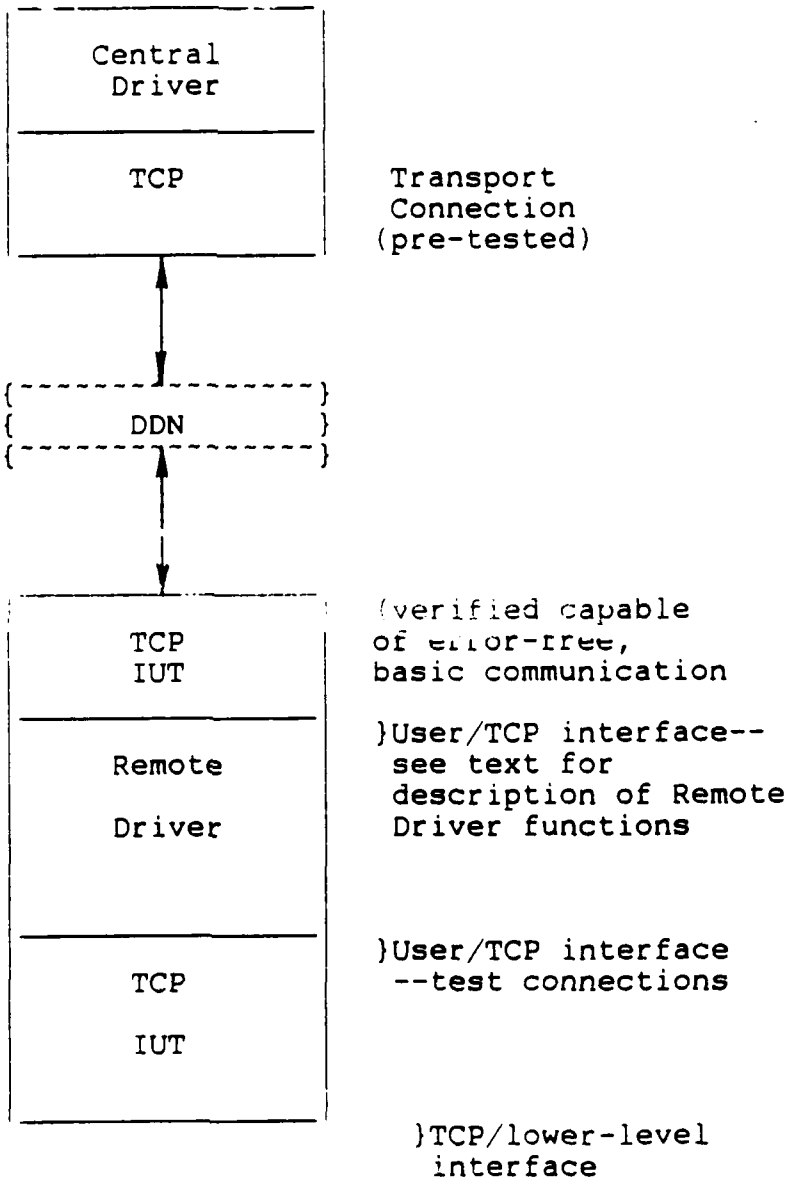
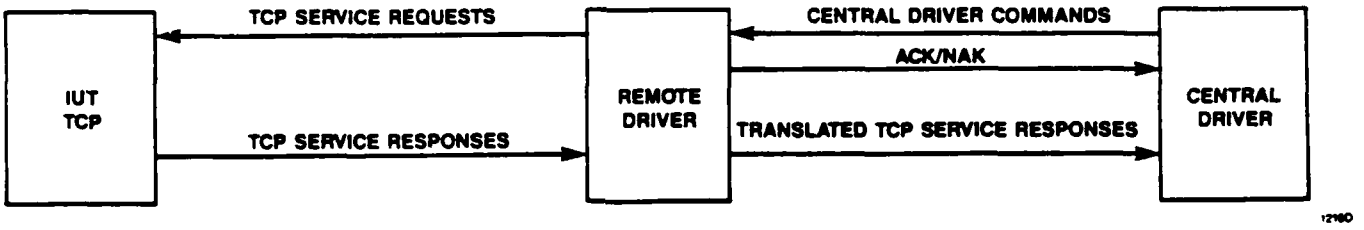


Figure 3.2-1. Remote Driver Functions

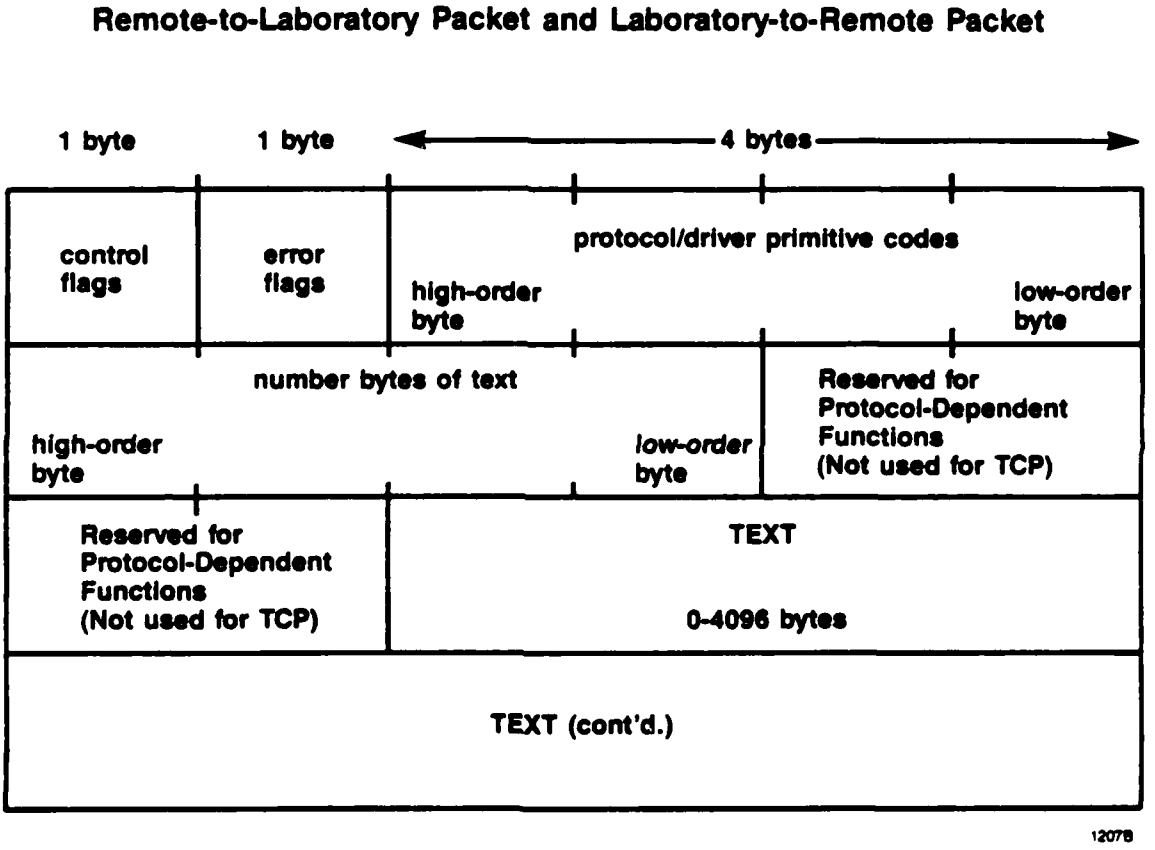


3.3 COMMAND CHANNEL COMMUNICATION

The Remote Driver must be able to receive packets from the Central Driver, translate number codes into commands, and interpret ASCII text strings of parameters. It also must be able to send one of three packets: an ACK (Acknowledgment), a NAK (Negative Acknowledgment), or a Data packet containing either protocol responses or driver command results. The Remote Driver is required to acknowledge the receipt of every driver or protocol command (ACK = positive acknowledgment; NAK = negative acknowledgment). When it is able to interpret a packet from the Central Driver, the Remote Driver sends an ACK packet. If it is unable to parse the packet command parameter list, the Remote Driver should send a NAK packet. A NAK is an indication of a Remote Driver problem and should be sent only during Remote Driver shakedown, never during the running of the TCP tests. In other error conditions, the Remote Driver sends one of the responses defined in Section 5, rather than a NAK packet.

All codes are explained in detail in the following sections. Figure 3.3-1 shows the structure of the packets on the command channel; Figure C.2-1 in Appendix C defines a packet in C syntax. The packet and each of its fields are ordered in Internet Protocol (IP) byte order.

Figure 3.3-1. Structure of the Packets on the Command Channel



3.3.1 Packet Control Flag Field

A single byte contains the control flag. (See Figure 3.3-1.) The bit positions are in ascending order from right to left. Figure 3.3.1-1 diagrams the bit-ordering scheme.

Figure 3.3.1-1. The Bit Order of a Byte

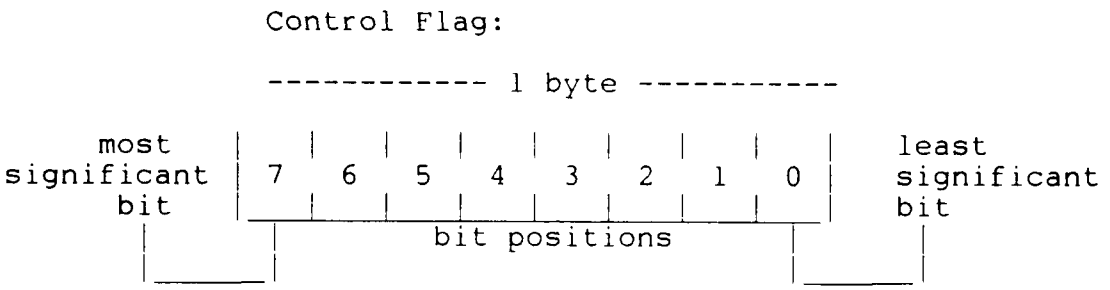


Table 3.3.1-1 summarizes the configuration of bit positions in the control flag. The three least significant bits are indicators that define how the driver is to interpret the packet. In the Remote Driver to Central Driver dialog, the Remote Driver sets the bit in position zero (0) to one (1) to indicate that an ACK packet is being sent. A zero (0) in this position indicates a NAK. If the Remote Driver sets the bit in position 1, then a data packet (as opposed to an ACK/NAK packet) is being sent. Setting the DATA bit in a packet could signal either that the Remote Driver is sending a protocol response, or that it is sending the results of one of its own driver operations. In the Central Driver to Remote Driver dialog, the bit in position 2 of the first byte of each packet sent by the Central Driver determines whether the packet contains a protocol or a driver command.

Table 3.3.1-1. Bit Positions in the Control Flag

a. Remote Driver --> Central Driver

Bit Position	ACK/NAK Packet	Data Packet
0	ACK = 1 NAK = 0	0 (unused)
1	0 (unused)	data = 1
2	0 (unused)	0 (unused)
3-7	0 (unused)	channel number

b. Central Driver --> Remote Driver

Bit Position	Command Packet
0	0 (unused)
1	0 (unused)
2	protocol command = 1 driver command = 0
3-7	channel number

Bits 3 through 7 form a test system channel number in the range 0 to 31. The channel number enables the Central Driver to sort Remote Driver data packets containing protocol responses before the Central Driver has interpreted the text of the packet. One channel number may be associated with more than one TCP test connection. Section 5.1.1.2 details the association between the channel number and TCP test connections.

3.3.2 Packet Error Code Field

This one-byte field is currently unused.

3.3.3 Packet Primitive Code Field

The 4-byte primitive code field specifies to the Remote Driver what action the Central Driver wants performed. This action can be one that exercises the TCP (a protocol primitive action) or one that changes the test environment or affects the driver itself (a driver primitive action). The interpretation is determined by the value of the second bit in the control flag field, as discussed in Paragraph 3.3.1. The driver must translate the integer in the primitive code to its corresponding protocol or driver primitive according to the descriptions and the code numbers in Tables 4.1-1 and 4.3-1. Section 4 gives a a detailed description of the primitive codes and their meanings.

3.3.4 Packet Num bytes Field

The next 4 bytes in the packet specify the number of bytes in the packet's text portion. This field can have a value of 0 to 4096, but the value must be 0 for an ACK or NAK packet. The Remote Driver must always specify the number of bytes in the text portion of a data packet.

3.3.5 Packet Reserved Field

The next 4 bytes in the packet are reserved. They are currently unused.

3.3.6 Packet Text Field

This field contains ASCII text. Its length is specified in the num_bytes field.

SECTION 4 - INTERPRETATION OF THE PRIMITIVE CODES

The primitive code field can be interpreted in two ways--either as a protocol primitive code or as a driver primitive code. A primitive in this sense means a command that describes some action within the protocol or the driver. The interpretation is determined by the value of the second bit in the control flag field, as discussed in section 3.3.1.

4.1 PROTOCOL PRIMITIVE CODES

If the control flag indicates a protocol command (i.e., the bit in position 2 is set to 1), then the Remote Driver must translate the integer in the primitive code field to its corresponding protocol primitive (Table 4.1-1). Because every operating system has different facilities, the form of commands given to the protocol IUT by the Remote Driver depends on the IUT's User Interface. The Remote Driver is required to include this translation capability in its function.

The Remote Driver can determine whether the required arguments are present by reading the packet num_bytes field, which tells how many bytes of ASCII character data are in the text field. If the num_bytes field contains a zero, then the Remote Driver assumes no arguments have been supplied. However, if the num_bytes field contains any positive integer from 1 to 4096, the Remote Driver must read the contents of the text field and interpret these contents as the primitive's arguments.

Arguments are located in the text field of the command packet--beginning at offset zero--and are separated by ASCII

spaces. The supplied arguments must be interpreted as position-dependent tokens. The data type each token represents is given in Table 4.1-2, "Internal Representation or ASCII Arguments." For example, if the Remote Driver receives a packet containing the protocol command ACT_OPEN--which requires the parameters SOURCE_PORT, DESTINATION_PORT, DESTINATION_ADDR, PRECEDENCE, SECURITY_LEN, CLASSIFICATION_LEVEL, PROTECTION_AUTHORITY, ULP_TIMEOUT, and ULP_TIMEOUT_ACTION--then the text field might contain the following:

5000 2000 7.0.0.2 0 0 0 0 5 0

where the ASCII data, separated by spaces, correspond to the argument tokens. The tokens SOURCE_PORT and DEST_PORT are 5000 and 2000, respectively. The DESTINATION_ADDR is of type Internet Address Format (A.B.C.D), so the data for this token are 7.0.0.2. The next four tokens have no corresponding data, but they must have place holders (zeros, separated by spaces) to indicate the absence of data or the false condition of a flag. The fifth argument, ULP_TIMEOUT, is 5, indicating number of seconds (according to Table 4.1-2). The final argument is zero. In this case, as the table shows, a zero represents a certain timeout action, not absence of data.

Table 4.1-1 lists the primitive commands and their respective number codes and argument fields. Table 4.1-2 gives the TCP data types for each protocol argument, accompanied by brief descriptions where needed.

Table 4.1-1. Protocol Primitive Commands, Number Codes, and Arguments

Primitive (**No.Code**)	Argument(s)	Section for Reference
ACT_OPEN (**1**)	SOURCE_PORT, DESTINATION_PORT, DESTINATION_ADDR, PRECEDENCE, SECURITY_LEN, CLASSIFICATION_LEVEL, PROTECTION_AUTHORITY, ULP_TIMEOUT, ULP_TIMEOUT_ACTION	4.1.1
SPEC_PSV (**2**)	SOURCE_PORT, DESTINATION_PORT, DESTINATION_ADDR, PRECEDENCE, SECURITY_LEN, CLASSIFICATION_LEVEL, PROTECTION_AUTHORITY, ULP_TIMEOUT, ULP_TIMEOUT_ACTION	4.1.2
PSV_OPEN (**3**)	SOURCE_PORT, PRECEDENCE, SECURITY_LEN, CLASSIFICATION_LEVEL, PROTECTION_AUTHORITY, ULP_TIMEOUT, ULP_TIMEOUT_ACTION	4.1.3
ACT_WDATA (**4**)	SOURCE_PORT, DESTINATION_PORT, DESTINATION_ADDR, PRECEDENCE, SECURITY_LEN, CLASSIFICATION_LEVEL, PROTECTION_AUTHORITY PUSH_FLAG, URGENT_FLAG, ULP_TIMEOUT, ULP_TIMEOUT_ACTION	4.1.4
SEND (**5**)	LCN, PUSH_FLAG, URGENT_FLAG, ULP_TIMEOUT, ULP_TIMEOUT_ACTION	4.1.5

Note: The arguments DATA and DATA_LEN are not used in the ACT_WDATA and SEND protocol commands because the driver command GEN_SND_TEXT (paragraph 4.3.2) always precedes these protocol commands.

(Table 4.1-1., cont'd.)

Primitive (**No.Code**)	Argument(s)	Section for Reference
ALLOC (**6**)	LCN, DATA_LEN	4.1.6
CLOSE (**7**)	LCN	4.1.7
ABORT (**8**)	LCN	4.1.8
STATUS (**9**)	LCN	4.1.9

Table 4.1-2. Definition of Command and Response Arguments

Argument	Type	Values
SOURCE_ADDR	Internet Address Format **A.B.C.D**	0 - 255. for **A,B,C** and **D.**
SOURCE_PORT	INTEGER	0 - 65535
DESTINATION_ADDR	Internet Address Format **A.B.C.D**	0 - 255. for **A,B,C** and **D.**
DESTINATION_PORT	INTEGER	0 - 65335.
LCN	INTEGER	0 - 65335. (This restriction is specific to the test system.)
ULP_TIMEOUT	INTEGER (seconds)	0 - 255.
ULP_TIMEOUT_ACTION	INTEGER	If ULP_TIMEOUT > 0, 1 for reset action, 0 for notify action. If ULP_TIMEOUT = 0, 0 for unused.
PRECEDENCE	INTEGER	0 - 7. If PRECEDENCE = 0, use the IUT default value.
SECURITY_LEN	INTEGER	0,4 or 5. 4 or 5 when CLASSIFICATION_LEVEL and PROTECTION_AUTHORITY are not 0, 0 when IUT default value is to be used. (This argument is specific to the test system. It designates the length of the option.)

(Table 4.1-2., cont'd.)

Argument	Type	Values
CLASSIFICATION_LEVEL	INTEGER	Valid security level or 0. 0 when IUT default value is to be used.
PROTECTION_AUTHORITY	INTEGER	Valid security protection authority or 0. 0 when IUT default value is to be used.
PUSH_FLAG	INTEGER	0 or 1. 0 when push service is not requested, 1 when push service is requested. Not used in responses.
URGENT_FLAG	INTEGER	0 or 1. 0 when data are not urgent, 1 when data are urgent.
DATA	ASCII	ASCII data.
DATA_LEN	INTEGER	Length of ASCII data (in bytes).
CONNECTION_STATE	ASCII	One of the following strings: LISTEN, SYN_SENT, SYN_RECEIVED, ESTABLISHED, FIN_WAIT1, FIN_WAIT2, CLOSE_WAIT, LAST_ACK, CLOSING, TIME_WAIT, or CLOSED.
SEND_WINDOW	INTEGER	0 - 65535.
RECEIVE_WINDOW	INTEGER	0 - 65535.
AMOUNT_OF_UNACKED_DATA	INTEGER	0 - 65535.
AMOUNT_OF_UNRECEIVED_DATA	INTEGER	0 - 65535.

(Table 4.1-2., cont'd.)

Argument	Type	Values
URGENT_STATE	INTEGER	0 or 1. 0 when the ULP is not in urgent mode, 1 when the ULP is in urgent mode. (This simple version of the STATUS URGENT_STATE field is specific to the test system.)

4.1.1 ACT_OPEN

Protocol command ACT_OPEN tells the Remote Driver to invoke the Active Open TCP service request primitive. This command allows an Upper Level Protocol (ULP) to initiate a connection attempt to a named ULP.

4.1.2 SPEC_PSV

Protocol command SPEC_PSV tells the Remote Driver to invoke the Fully Specified Passive Open TCP service request primitive. This command allows a ULP to listen for and respond to connection attempts from a fully named ULP.

4.1.3 PSV_OPEN

Protocol command PSV_OPEN tells the Remote Driver to invoke the Unspecified Passive Open TCP service request primitive. This command allows a ULP to listen for and respond to connection attempts from an unnamed ULP.

If, after a connection has been established on the listening port, the Central Driver wishes the Remote Driver to invoke another Unspecified Passive Open TCP service request on the same port, another PSV_OPEN protocol command will be sent. Remote Driver implementors should be aware that in UNIX-based systems this passive open service request will have already been done automatically, and should provide for handling this second request in their driver implementations.

4.1.4 **ACT WDATA**

Protocol command ACT_WDATA tells the Remote Driver to invoke the Active Open With Data TCP service request primitive. This command allows a ULP to initiate a connection attempt to a named ULP accompanied by the specified data. ACT_WDATA is always preceded by the driver command GEN_SND_TXT documented in section 4.3.2.

4.1.5 **SEND**

Protocol command SEND tells the Remote Driver to invoke the Send TCP service request primitive. This command causes data to be transferred across the named connection. SEND is always preceded by the driver command GEN_SND_TXT documented in section 4.3.2.

4.1.6 **ALLOC**

Protocol command ALLOC tells the Remote Driver to invoke the Allocate TCP service request primitive. This command allows a ULP to issue TCP an incremental allocation for receiving data. The parameter DATA_LEN is defined in single-byte units. This quantity is the additional number of bytes the receiving ULP is willing to accept.

NOTE: The ALLOC protocol command corresponds exactly to the MIL-STD-1788 service request. The Remote Driver issues this service request at its own convenience unless it is in the Explicit mode (Section 4.2).

4.1.7 **CLOSE**

Protocol command CLOSE tells the Remote Driver to invoke the Close TCP service request primitive. This command allows a ULP to indicate that it has completed data transfer across the named connection.

4.1.8 **ABORT**

Protocol command ABORT tells the Remote Driver to invoke the Abort TCP service request primitive. This command allows a ULP to indicate that the named connection is to be immediately terminated.

4.1.9 **STATUS**

Protocol command STATUS tells the Remote Driver to invoke the Status TCP service request primitive. This command allows a ULP to query for the current status of the named connection.

4.2 REMOTE DRIVER MODES

The Remote Driver is required to have two modes of receiving data across its TCP test connections: the Implicit and the Explicit modes. The Implicit mode is the default mode.

4.2.1 Implicit Mode

In the default, the Implicit mode, the Remote Driver is responsible for making sure the TCP IUT can deliver data. Depending on the specific test, varying amounts of data will arrive on the test connections, and the Remote Driver must be prepared for a maximum volume similar to that of file transfer. In terms of the TCP specification, the Remote Driver must issue suitable Allocate service requests. It does not receive any Allocate protocol commands from the Central Driver.

4.2.2 Explicit Mode

In the Explicit mode, the Remote Driver may not issue Allocate service requests unless it receives the protocol command from the Central Driver. The Remote Driver enters this mode when it receives an EXPLICIT driver command and leaves it when it receives an END_EXPLICIT driver command (Section 4.3).

4.3 DRIVER PRIMITIVE CODES

If the control flag indicates a driver command (the bit in position 2 is set to zero), the Remote Driver must translate the integer contained in the code field to its corresponding driver command (Table 4.3-1). The Remote Driver then performs the appropriate action, as specified in the following sections.

Table 4.3-1. The Driver Primitives

Code Number	Remote Driver Action (Arguments in parentheses; if none, then (---))	Section
0	**KILL** (---) Kill the Remote Driver process.	4.3.1
1	**GEN_SND_TEXT** (TEXT_CHR, TEXT_LEN) Generate text to be used as the data in a SEND or ACT_WDATA protocol command.	4.3.2
2	**EXPLICIT** (---)	4.3.3
3	**END_EXPLICIT** (---)	4.3.4
4	**NOOP** (---) No operation.	4.3.5

4.3.1 KILL

If it interprets a driver primitive code of 0, the Remote Driver ACKs the driver primitive packet. Then the Remote Driver aborts all test connections, including the command channel connection, and terminates itself.

4.3.2 GEN_SND_TEXT

If the Remote Driver receives a driver primitive code of 1, indicating the GEN_SND_TEXT command, then after ACKing the driver primitive packet, the Remote Driver must generate text specified by the arguments TEXT_CHR (a single ASCII character) and TEXT_LEN (integer). The arguments should be interpreted as tokens, as described in Section 4.1. A

single character (TEXT_CHR) is generated the specified number (TEXT_LEN) of times, and is subsequently used in either the next SEND or ACT_WDATA protocol command. Either one of these protocol commands always immediately follows the GEN_SND_TEXT driver command.

4.3.3 EXPLICIT

If the Remote Driver receives a driver primitive of 2, it should enter the EXPLICIT mode. Thereafter, it may not issue Allocate service requests unless it receives the protocol command ALLOC from the Central Driver (Section 4.2.2).

4.3.4 END_EXPLICIT

On receipt of the driver primitive code 3, the END_EXPLICIT command, the Remote Driver returns to the IMPLICIT mode. It may issue suitable Allocate service requests as needed (Section 4.2.1).

4.3.5 NOOP

If the Remote Driver receives a driver primitive code of 4, indicating the NOOP command, the only action required of the Remote Driver is that it ACK the packet. The test system uses the NOOP primitive to verify communications capability and to conduct Are-You-There checks.

SECTION 5 - REMOTE DRIVER RESPONSES

The Remote Driver sends two types of packets to the Central Driver: ACK/NAK packets and Data packets. It must acknowledge all command packets from the Central Driver with an ACK or NAK packet and must send to the Central Driver all protocol responses it receives from the TCP IUT. The protocol responses are reported in data packets with the packet header specifying the responding test channel and the text portion indicating the response. Section 3 gives a detailed description of the packet format. Figure 5-1 shows an example of a Data packet from the Remote Driver to the Central Driver. Appendix D contains an example sequence of packet exchanges between a Remote Driver and a Central Driver.

5.1 PACKET HEADER

5.1.1 Control Flag

5.1.1.1 Type of Packet Designation

For an ACK/NAK packet, the Remote Driver sets bit position 0 in the Control Flag byte: 1 indicates an ACK; 0 specifies a NAK. Bit positions 1 and 2 are set to 0. In a Data packet, bit position 0 is set to 0, bit position 1 is set to 1 (indicating a Data packet), and bit position 2 is set to 0.

5.1.1.2 Channel Designation

The Remote Driver must set the channel number in bits 3 through 7 in the control flag of all packets it sends to the Central Driver. It sets the channel number to 0 on ACK/NAK packets. The Remote Driver must set the field for

Figure 5-1. Example Command Channel Packet -- Remote Driver to Central Driver

CONTROL FLAGS	ERROR FLAGS	PRIMCODE			
00010010	0 0	0 0	0 0	0 0	0 0
NUMBER OF BYTES OF TEXT				RESERVED	
0 0	0 0	0 0	1 4	0 0	0 0
(RESERVED)		TEXT			
0 0	0 0	O	P	E	N
ø	I	D	ø	5	ø
2	0	1	0	ø	ø
ø	E	N	D		

EXPLANATION

CONTROL FLAGS: DATA PACKET
CHANNEL 2

NUMBER OF BYTES OF TEXT: 14 hexadecimal (20 bytes)

TEXT: Open ID response to a PSV_OPEN protocol command. Destination port and address not known, so two blanks are substituted. LCN is 5. Source port is 2010. The ASCII "END" is the end of message delimiter.

ø - This symbol is the ASCII equivalent of the space character.

Data packets to the channel number associated with the connection that received the reported response. Normally, one channel number is associated with each TCP test connection (each LCN). One channel number is associated with more than one LCN during tests of IUT maintenance of more than 32 connections. The channel number on the protocol command packet that causes the Remote Driver to issue an open command (ACT_OPEN, SPEC_PSV, PSV_OPEN) is associated with the TCP test connection/LCN that results for the lifetime of the connection.

The Remote Driver must also set a channel number in Data packets containing protocol responses that do not refer to a TCP test connection. These are the Open Failure and TCP Error responses (the latter only in some cases) and also the System Error driver response. In these cases, the Remote Driver uses the channel number from the command packet containing the protocol command that produced the response.

5.1.2 Num bytes Field

The num_bytes header field must contain the number of bytes in the text portion of the packet. This field is always set to 0 for an ACK/NAK packet. A response Data packet will always have a text portion, and this field must be set to the length of the text.

5.2 TEXT PORTION--PROTOCOL RESPONSES

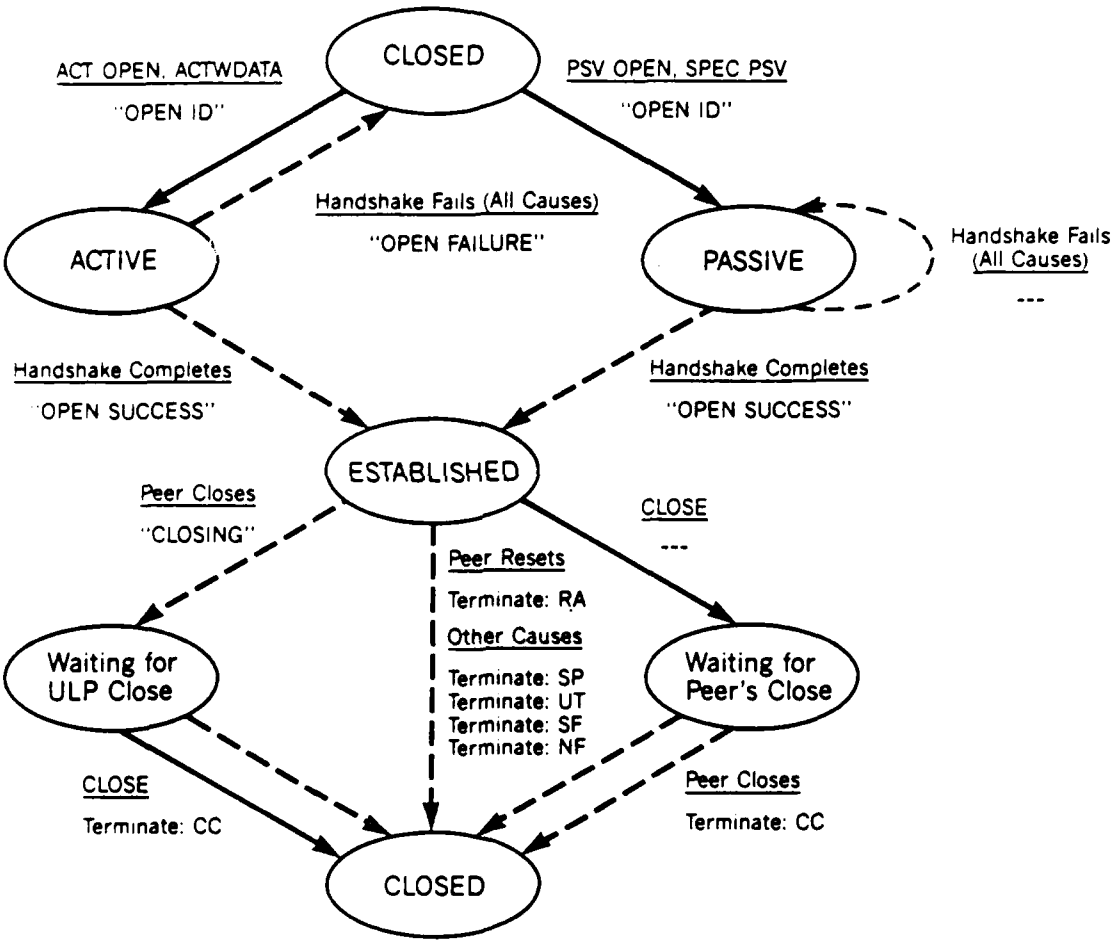
The text portion of the response Data packet must contain the responses from the IUT TCP or driver responses (5.3). MIL-STD-1778 defines the service response primitives

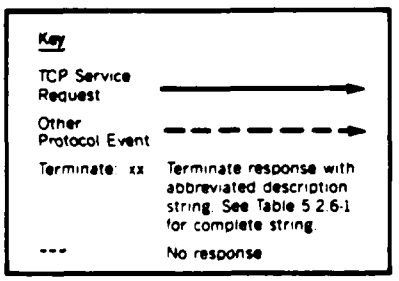
that TCP provides through its service response interface. The Remote Driver must convert whatever form of the primitives it receives from its TCP into Remote Driver protocol responses made up of the ASCII strings specified in the following sections. A guide to this conversion is given in the Remote Driver Service State Machine (Figure 5.2-1). The convention applied to the descriptions of Remote Driver protocol responses is:

- All ASCII characters or symbols appearing outside angle brackets ("< >") must appear in the string to be sent;
- All strings placed inside angle brackets are tokens to be replaced by an ASCII representation of the actual value or the default (the TCP data types of these arguments are shown in Table 4.1-2);
- One ASCII space must occur between each token; and
- Line breaks are shown only for clarity--they should be ignored.

Figure 5.2-1. Remote Driver Service State Machine

rdspec.fsm





Additional Transitions

State:	ALL STATES	ACTIVE or PASSIVE STATE
Event:	ABORT	CLOSE
Action:	Terminate: UA	Terminate: CC
Next State:	CLOSED	CLOSED

1858E—46JJ
05/27/87

5.2.1 The **Open Id** Response

The Remote Driver sends this protocol response to the Central Driver when it receives an Open Id service response from its TCP. An Open Id protocol response indicates the local connection name assigned by TCP to the connection requested in one of the previous service requests--Unspecified Open, Fully Specified Open, or Active Open.

OPEN ID <LCN> <SOURCE_PORT>
<DESTINATION_PORT> (if known)
<DESTINATION_ADDR> (if known) **END**

(Supply 1 space for unknown arguments)

5.2.2 The **Open Failure** Response

The Remote Driver sends this protocol response to the Central Driver when it receives an Open Failure service response from its TCP. This protocol response indicates the failure of an Active Open service request.

OPEN FAILURE <LCN> **END**

5.2.3 The **Open Success** Response

The Remote Driver sends this protocol response to the Central Driver when it receives an Open Success service response from its TCP. The Open Success protocol response indicates the successful completion of one of the Open service requests.

OPEN SUCCESS <LCN> **END**

5.2.4 The **Deliver** Response

The Remote Driver sends this protocol response to the Central Driver when it receives a Deliver service response from its TCP. The Deliver protocol response indicates the arrival of data across the named connection.

DELIVER <LCN> <DATA_LEN> <DATA>
URG=<URGENT_FLAG> **END**

Note: No spaces exist on either side of the equal sign ("=").

5.2.5 **Closing** Response

The Remote Driver sends this protocol response to the Central Driver when it receives a Closing service response from its TCP. The Closing protocol response indicates that the peer ULP has issued a CLOSE service request and that TCP has delivered all data sent by the remote ULP. This response is issued only if the local ULP has not yet sent a Close. (See Figure 5.2-1.)

CLOSING <LCN> **END**

5.2.6 **Terminate** Response

The Remote Driver sends this protocol response to the Central Driver when it receives a Terminate service response from its TCP. The Terminate protocol response indicates that the named connection has been terminated and no longer

exists. TCP generates this response as a result of a remote connection reset, service failure, or connection closing by the local ULP.

TERMINATE <LCN> : <DESCRIPTION> **END**

where the colon (":") has one space on either side and "<DESCRIPTION>" is one of the following required description strings:

REMOTE ABORT
NETWORK FAILURE
SEC/PREC MISMATCH
ULP TIMEOUT
ULP ABORT or USER ABORT*
ULP CLOSE or CONNECTION CLOSED*
SERVICE FAILURE

* MIL-STD-1778 allows either (sections 6.5.6.1 and 9.4.6.3.34).

Table 5.2.6-1 provides an explanation of these description strings.

Table 5.2.6-1. Meanings of Terminate Description Strings

Description String	Meaning
REMOTE ABORT	Remote abort.
NETWORK FAILURE	Network failure.
SEC/PREC MISMATCH	Received segment has unmatched security or precedence.
ULP TIMEOUT	Expiration of a ULP timeout when timeout action is reset.
ULP ABORT or USER ABORT	Abort issued by local ULP.
ULP CLOSE or CONNECTION CLOSED	Close issued by the local ULP **after the remote TCP has closed** or Arrival of remote TCP's close **after the local ULP has issued a close.**
SERVICE FAILURE	Abort by the local TCP for other reasons (service failure).

5.2.7 **Status** Response

The Remote Driver sends this protocol response to the Central Driver when it receives a Status service response from its TCP. The Status protocol response indicates the current status information associated with a connection named in a previous Status service request.

STATUS <LCN> <SOURCE_PORT>
<SOURCE_ADDR> <DESTINATION_PORT>
<DESTINATION_ADDR>
<CONNECTION_STATE> <SEND_WINDOW>
<RECEIVE_WINDOW>
<AMOUNT_OF_UNACKED_DATA>
<AMOUNT_OF_UNRECEIVED_DATA>
<URGENT_STATE> <PRECEDENCE>
<CLASSIFICATION_LEVEL>
<PROTECTION_AUTHORITY>
<ULP_TIMEOUT> <ULP_TIMEOUT_ACTION>
END

5.2.8 **TCP Error** Response

The Remote Driver sends this protocol response to the Central Driver when it receives an Error service response from its TCP. The TCP Error response indicates illegal service requests relating to the named connection or errors relating to the environment.

TCP ERROR <LCN> : <ERROR DESCRIPTION> **END**

(If the LCN argument is not known, supply 1 space.) The colon (":") has a space on either side, and "<ERROR DESCRIPTION>" is one of the following required description strings:

INSUFFICIENT RESOURCES
SEC/PREC NOT ALLOWED
CONNECTION DOES NOT EXIST
ILLEGAL REQUEST
CONNECTION ALREADY EXISTS
CONNECTION CLOSING
ULP TIMEOUT
NO MESSAGE

Note: The ASCII-formatted protocol responses described above correspond to primitives required by MIL-STD-1778. Section 6.4.10 in MIL-STD-1778 describes these responses, and sections 6.5.6.1 and 9.4.6 give a full specification of TCP events and actions requiring these responses.

5.3 TEXT PORTION--DRIVER RESPONSES

The text portion of the response Data packet may contain a driver response. The Remote Driver sends a driver response to indicate an error that affects the performance of a TCP command, but that is not one of the error or failure conditions defined by the TCP specification.

5.3.1 The **System Error** Response

The Remote Driver sends this response to inform the Central Driver of system-dependent problems affecting the performance of a TCP command:

SYSTEM ERROR : ‹DESCRIPTION› **END**

Where the colon (":") has one space on either side and "<DESCRIPTION>" is one of the following description strings:

REQUESTED SOURCE PORT NOT PERMITTED
REQUESTED SOURCE PORT IN USE
REQUESTED SERVICE NOT IMPLEMENTED
REQUESTED PARAMETER NOT IMPLEMENTED

SECTION 6 - TIMING

The Remote Driver should be started up before testing begins. All tests of TCP require the maintenance of at least two TCP connections: the command channel from/to the Central Driver and one or more test connections. Because the next command from the Central Driver may depend on the timely arrival of a Remote Driver protocol response, the Remote Driver must not be blocked from receiving commands from the Central Driver in preference to data and service responses from the test connections.

It is recommended that the Remote Driver check frequently for data. The amount to receive at each check of a test connection depends on the constraints of the operating system, but the goal is to handle a maximum volume similar to that of file transfer. Apart from these points, a Remote Driver has no intrinsic timing constraints, but it should not add considerably to a protocol IUT's response time.

SECTION 7 - FLEXIBILITY

Although drivers have no special flexibility requirements, adaptable hardware and software enhance their operation and expandability. In the next phase of protocol testing, drivers may need to be capable of commanding passive recorders or other devices.

APPENDIX A - References

"Military Standard Transmission Control Protocol" MIL-STD-1778); August 1983; Department of Defense.

"Military Standard Internet Protocol" (MIL-STD-1777); August 1983; Department of Defense.

System Development Corporation, "Laboratory Implementation Plan," TM-WD-8574/000/02, January 1985.

System Development Corporation, "Laboratory Specification," TM-WD-7172/520/00, August 1984.

System Development Corporation, "Higher Level Capability Plan," TM-WD-8573/000/00, April 1984.

Kernighan, B. W., and Ritchie, D. M.; The C Programming Language; Prentice-Hall, Inc.; Englewood Cliffs, NJ; 1978.

Kernighan, B. W., and Pike, R.; The UNIX Programming Environment; Prentice-Hall, Inc.; Englewood Cliffs, NJ; 1984.

Leffler, S. J., Fabry, R.S., and Joy, W.N.; "A 4.2 BSD Interprocess Communication Primer"; Computer Systems Research Group, Department of Electrical Engineering and Computer Science, University of California, Berkeley; 1985.

APPENDIX B - Glossary

ACK (Acknowledgment)

In data transfer between devices, data are blocked into units of a size given in each block's header. If the received data are found to be without errors, then the receiving device sends an ACK block back to the transmitting unit to acknowledge receipt. The transmitting device then sends the next block. If the receiving unit detects errors, however, it sends a NAK block to indicate the received data contained errors.

ASCII (American Standard Code for Information Interchange)

A standard code for the representation of alphanumeric information. ASCII is an 8-bit code in which 7 bits indicate the character represented and the 8th, high-order bit is used for parity.

DEC (Digital Equipment Corporation)

IUT (Implementation Under Test)

A protocol implementation that is the subject of the immediate test.

MIL-STD (Military Standard)

Specification published by the DoD.

NAK (Negative Acknowledgment)

In data transfer between devices, a NAK block is returned by the receiving device to the sending device to indicate the preceding data block contained errors. See also ACK.

packet switching

A method of transmitting messages through a network in which long messages are subdivided into short packets. The packets are then transmitted as in message switching.

PAD (Packet Assembler/Disassembler)

In this document, PAD refers to a module of a structured program responsible for reading and writing data packets. Not to be confused with the other known usage, which describes a device to provide service to asynchronous terminals within an X.25 network.

PAR (Positive Acknowledgment and Response)

A simple communication protocol stating that every packet received must be either ACKed or NAKed.

protocol

A set of rules governing the operation of functional units to achieve communication.

TCP (Transmission Control Protocol)

The DoD standard connection-oriented transport protocol used to provide reliable, sequenced, end-to-end service.

ULP (Upper Level Protocol)

Any protocol above TCP in the layered protocol hierarchy that uses TCP. This term includes presentation layer protocols, session layer protocols, and user applications such as the Protocol Test System's drivers.

APPENDIX C - Examples of Remote Driver Implementation in UNIX/C

C.1 CONNECTION ESTABLISHMENT

The Protocol Test System runs in a UNIX 4.2 BSD environment. The example Remote Driver is implemented in **C** language, which provides access to several interprocess communication system calls (e.g., the **fork**, **socket**, **bind**, **listen**, and **accept** system calls).

The design of the Remote Driver uses the Internet server model. The first step taken by the Remote Driver is to disassociate itself from the controlling terminal of the invoker. It then does a passive open and listens at the published well-known port for a connection request from the Surrogate Driver. Using the fork system call, the Remote Driver creates a copy of itself (a child process), so that it can continue to listen for connection requests following a successful connection. It is this child process that carries out the functions of the Remote Driver, communicating over the established command channel. The original or parent process remains in execution and listens to the well-known port. Figure C.1-1 outlines the establishment of the command channel in 4.2 BSD UNIX/C. In addition to utilizing the system calls, the example uses the 4.2 BSD library calls to eliminate direct handling of Internet protocol numbers and addressing.

Figure C.1-1. Outline of Command Channel Establishment in 4.2 BSD UNIX C

```
 *                    COMMAND CHANNEL ESTABLISHMENT                    */
struct protoenet *pp;
struct servent *sp;
pp = getprotobyname("tcp");    * 4.2 BSD library call */
sp = getservbyname("remotedriver","tcp"); /* 4.2 BSD library
call */
#ifndef DEBUG
     <<disassociate remote driver from controlling
     terminal>>#endif
     ...
     sin.sin_port = sp->s_port;
     ...
     s = socket(AF_INET, SOCK_STREAM, pp->p_proto);
     bind(s, (caddr_t)&sin, sizeof(sin), 0);
     listen(s, 5); /* passive open to listen at well-known
     port */

     for(;;) {
           s2 = accept(s, 0, 0);

           if (fork() == 0) { /* this is the child */
           {
               close(s);
               do_remote_driver_functions(s2); /* s2 is the
local connection name of the command channel */
     }
           close(s2); /* the parent goes back to listening
on s */

     }
```

C.2 A PAD FUNCTION

A Packet Assembler/Disassembler (PAD) function is useful for implementing a Remote Driver. Because the two basic functions of receiving and transmitting packets are performed repeatedly, it may be wise to modularize them. When the Remote Driver reads data from the command channel, it must be able to interpret the bytes correctly. In a UNIX/C implementation, the method used to achieve this result is to load the data into a data structure where distinct fields can be declared. In C, this is the **struct** declaration. Each collection of fields can be declared and referenced as a whole. The term "packet" has been used in this document as the name of this data structure reference. The **struct** declaration is shown in Figure C.2-1.

Figure C.2-1. The C Syntax Format of the Data Packet

```
#define  MAX_TEXT_LEN       4096
struct remote_pack  {
              char cntl_flag;
              char err_flag;
              int  code;
              int  num_bytes;
              int  reserved;
              char text[MAX_TEXT_LEN];
            };
```

The reception mode of the PAD function reads data and "packets" the data. The PAD accomplishes this task by reading the first 14 bytes of data from the input stream. This first 14 bytes is effectively the header of the data packet. The header contains all the information needed to

process the packet, including the integer in the "num_bytes" field that indicates the number of bytes of character text to follow, if any. If the input data stream is correctly packeted into a data structure, then interpretation of the fields in the packet should become clearer and less susceptible to error.

The transmission mode of the PAD function unpackets the data and sends data over the communication channel. The PAD accomplishes this task by writing a packet header and then the text into a memory space allocated for a large character string. The size is equal to 4096 bytes--the maximum text size--plus 14 bytes for the header information. If the text size is less than the maximum, then only that much need be written. The Remote Driver then transmits the data as a normal byte stream.

An example of a PAD implemented in C is given in Figure C.2-2:

- The example is specific to UNIX/C on a VAX having 32-bit integers;
- The routines for converting between (VAX) word order are mandatory.

Figure C.2-2. A Packet Assembler/Disassembler in C

```
#define HEADER_SIZE   14
#define MAX_TEXT_LEN  4096
#define MAX_PACK_LEN HEADER_SIZE+MAX_TEXT_LEN

send_packet(packet,sock)      /* packet disassembler     */
struct remote_pack *packet;
int sock;
{    register int i;
     char send_buffer[MAX_PACK_LEN];

     send_buffer[0] = packet->cntl_flag;
     send_buffer[1] = packet->err_flag;    /* host-network
conversion */
     *((int *)(send_buffer+2))     = htonl(packet->code);
     *((int *)(send_buffer+6))     = htonl(packet->
                                      num_bytes);
     *((int *)(send_buffer+10))    = htonl(packet->reserved);
     for(i=0; i < packet->num_bytes; i++)
          send_buffer[i+14] = packet->text[i];
                              /* send the packet         */
     return(write(sock, send_buffer,
                  packet->num_bytes+HEADER_SIZE));
}

recv_packet(packet,sock)  /* packet assembler       */
struct remote_pack *packet;
int sock;
{
     char recv_buffer[HEADER_SIZE];
     int result;
                           /* read the header    */
     if    ( (result = read(sock, recv_buffer, HEADER_SIZE))
                           != HEADER_SIZE)
     {
            return (result);
     }
     packet->cntl_flag = recv_buffer[0];
     packet->err_flag  = recv_buffer[1];   /* net-host
conversion */
     packet->code      = ntohl(*((int *) (recv_buffer+2)));
     packet->num_bytes = ntohl(*((int *) (recv_buffer+6)));
     packet->reserved  = ntohl(*((int *) (recv_buffer+10)));
                           /* read the text      */
     if (packet->num_bytes)
     {
       if(read(sock, packet->text, packet->num_bytes)  !=
                                          packet->num_bytes)
          return(-1);
     }
     return(0);
}
```

APPENDIX D - Example Command Channel Packet Exchange Between the Central Driver (CD) and the Remote Driver (RD)

	PACKET FIELDS	EXPLANATION
CD to RD	CTRLFLAGS: 00010 100	Protocol Command, Channel is 2
	PRIMCODE: 3	Command is PSV_OPEN
	TEXT: **2010 3 0 0 0 0 0**	Arguments: Source Port is 2010, Precedence is 3, other arguments have default values.
RD to CD	CTRLFLAGS: 00000 001	ACK previous packet.
RD to CD	CTRLFLAGS: 00010 010	Data Packet (Protocol Response), Channel 2
	TEXT: **OPEN ID 5 2010 END**	Open Id response, the destination port and address are not known, so two blanks appear. LCN is 5, Source Port is 2010.

~ ~

PEER EXCHANGE BETWEEN TCP IUT AND TCP REFERENCE IMPLEMENTATION LEADS TO ESTABLISHMENT OF CONNECTION.

~ ~

RD to CD	CTRLFLAGS: 00010 010	Data Packet (Protocol Response), Channel 2
	TEXT: **OPEN SUCCESS 5 END**	Open Success response, LCN 5.
CD to RD	CTRLFLAGS: 00010 000	Driver Command, Channel 2
	PRIMCODE: 1	Command is GEN_SND_TEXT
	TEXT: **F 8**	Arguments: TEXT_CHR is "F", TEXT_LEN is 8.
RD to CD	CTRLFLAGS: 00000 001	ACK previous packet.
CD to RD	CTRLFLAGS: 00010 100	Protocol Command, Channel 2
	PRIMCODE: 5	Command is Send

TEXT: **5 1 0 0 0** Arguments: LCN 5, Push is on, Urgent is off, no ULP Timeout.

RD to CD CTRLFLAGS: 00000 001 ACK previous packet.

NOTE: There is no Driver Response to the Send Protocol Command. After acknowledgment, the previously generated data is sent over the indicated connection.

CD to RD CTRLFLAGS: 00010 100 Protocol Command, Channel 2

PRIMCODE: 7 Command is Close

TEXT: **5** Argument: LCN 5.

NOTE: There is no Driver Response to the Close Protocol Command until the TCP IUT informs the RD of the completion of the TCP closing handshake. Some time passes.

~ ~

PEER EXCHANGE BETWEEN TCP IUT AND TCP REFERENCE IMPLEMENTATION: TCP IUT CAN COMPLETE USER'S CLOSE, WHEN PEER HAS CLOSED.

~ ~

RD to CD CTRLFLAGS: 00010 010 Data Packet (Protocol Response), Channel 2

TEXT: **TERMINATE 5 : CONNECTION CLOSED END**

Terminate response, LCN 5, indicating completion of the TCP closing handshake.

NOTE: The association between Channel 2 and LCN 5 ends now because the connection ceases to exist. The next four packets show the exchange in a test of the TCP IUT's handling of an erroneous command to send data on a non-existent connection.

CD to RD CTRLFLAGS: 00010 000 Driver Command, Channel 2

PRIMCODE: 1 Command is GEN_SND_TEXT

TEXT: **F 8** Arguments: TEXT_CHR is "F", TEXT_LEN is 8.

RD to CD	CTRLFLAGS:	00000 001	ACK previous packet.
CD to RD	CTRLFLAGS:	00010 100	Protocol Command, Channel 2
	PRIMCODE:	5	Command is Send
	TEXT:	**5 1 0 0 0**	Arguments: LCN 5, Push is on, Urgent is off, no ULP Timeout.
RD to CD	CTRLFLAGS:	00000 001	ACK previous packet.

NOTE: The Driver ACKs the Send command packet even though the command is in error.

RD to CD	CTRLFLAGS:	00010 010	Data Packet (Protocol Response), Channel is 2 because Command packet evoking this response has Channel Number of 2 and not because of the LCN argument of the erroneous command.
	TEXT:	**TCP ERROR : CONNECTION DOES NOT EXIST END**	TCP Error response. There is no LCN, so one blank is substituted.